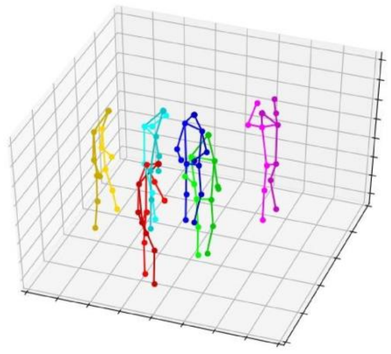
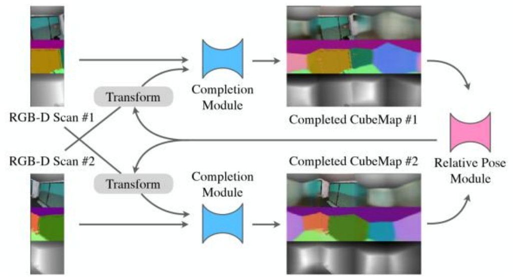
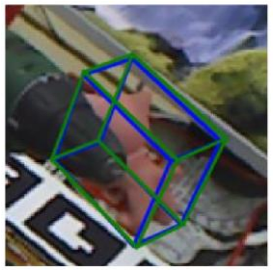
CS376 Computer Vision
Lecture 19: Machine Learning Basics

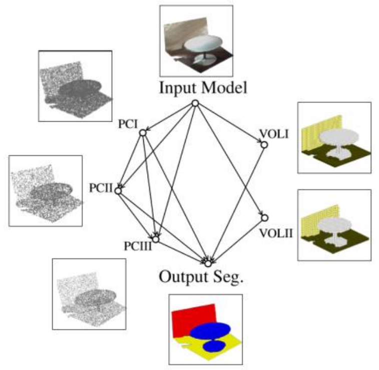




Qixing Huang

April 8th 2019





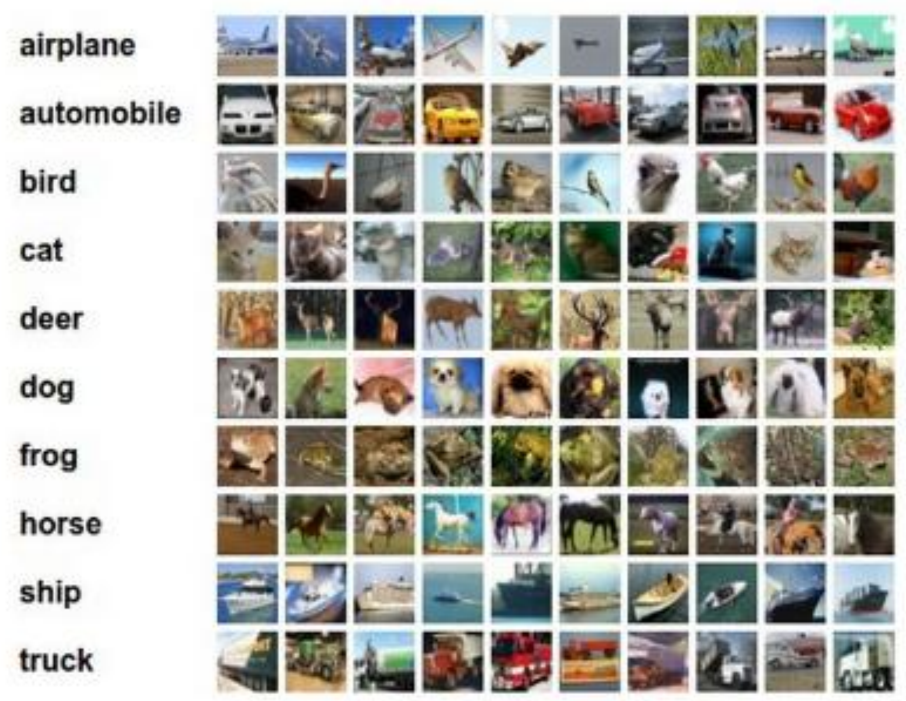
Slide material from: http://cs231n.github.io/classification/
http://cs231n.github.io/linear-classify/

Today’s Topic

- Nearest Neighbor
  - K-Nearest Neighbor Classifier
- Linear Support Vector Machine (SVM)

- Not covered in this class but later:
  - Adaboost
  - Random Forest

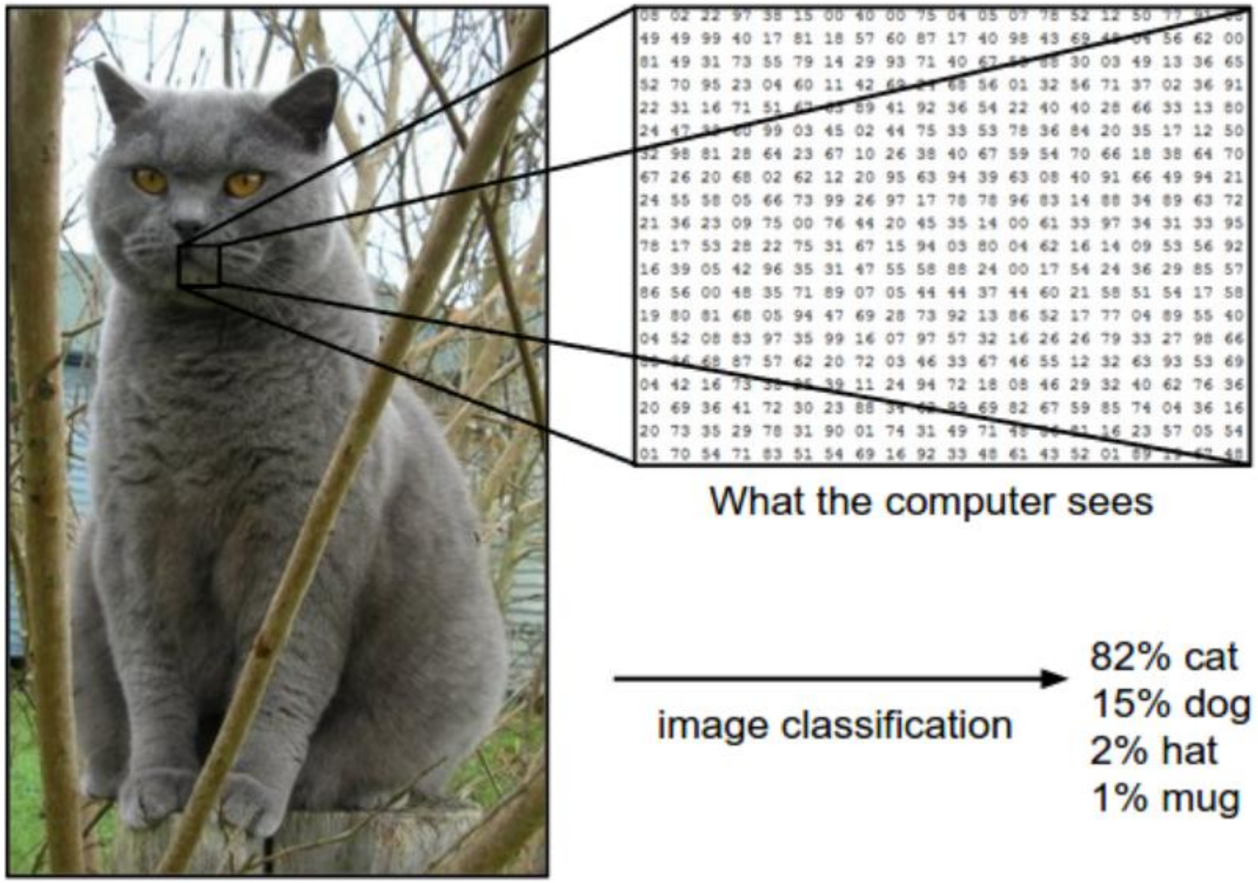
Image classification as the example



Cifar-10

We will cover more advanced topics in image classification later

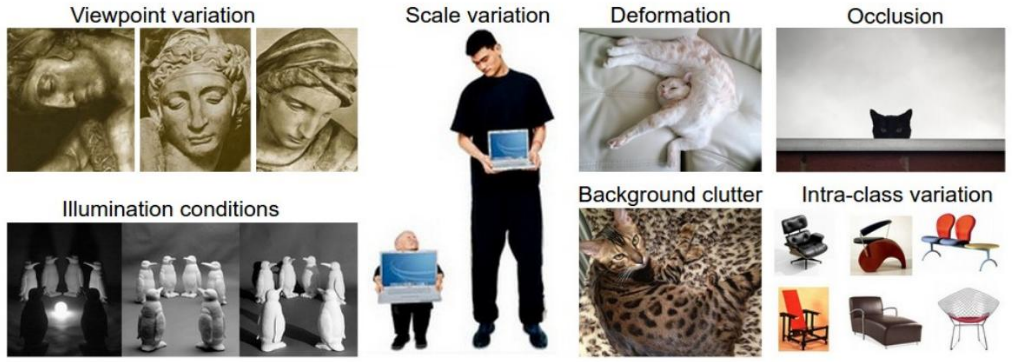
Image classification



The task in Image Classification is to predict a single label (or a distribution over labels as shown here to indicate our confidence) for a given image.

Challenges

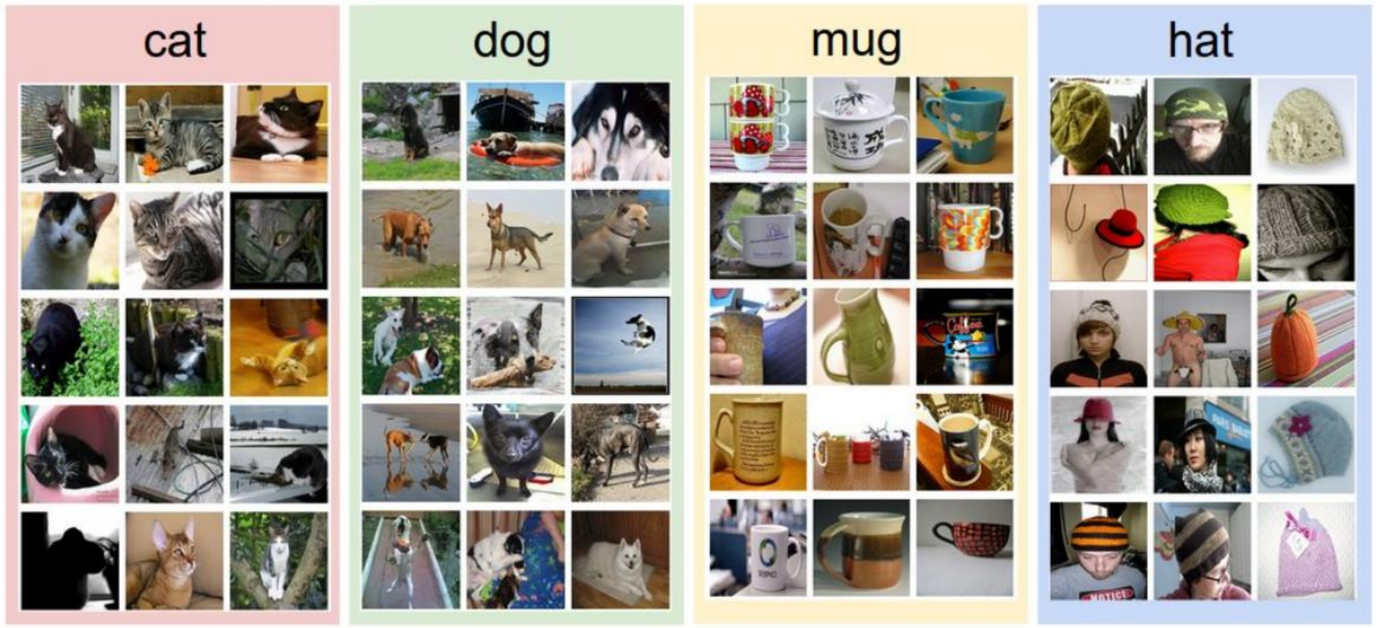
- Recall the challenges for computing image descriptors
- **Viewpoint variation**.
- **Scale variation**
- **Deformation**
- **Occlusion**
- **Illumination conditions**
- **Background clutter**
- **Intra-class variation**



A good image classification model must be invariant to the cross product of all these variations, while simultaneously retaining sensitivity to the inter-class variations.

Data-driven approach

- Instead of hand-coding each category
- Ask a computer to learn from data

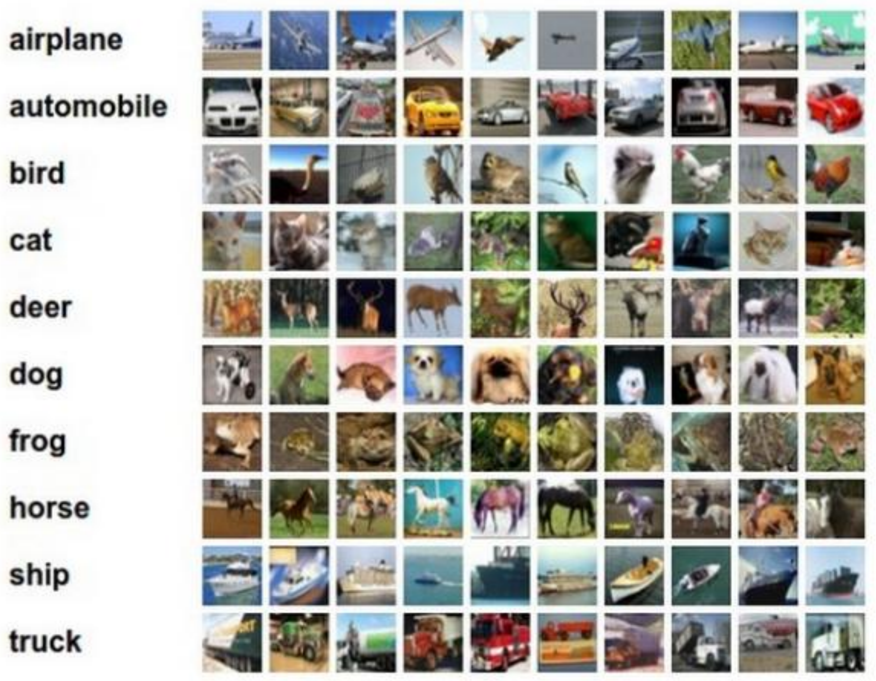


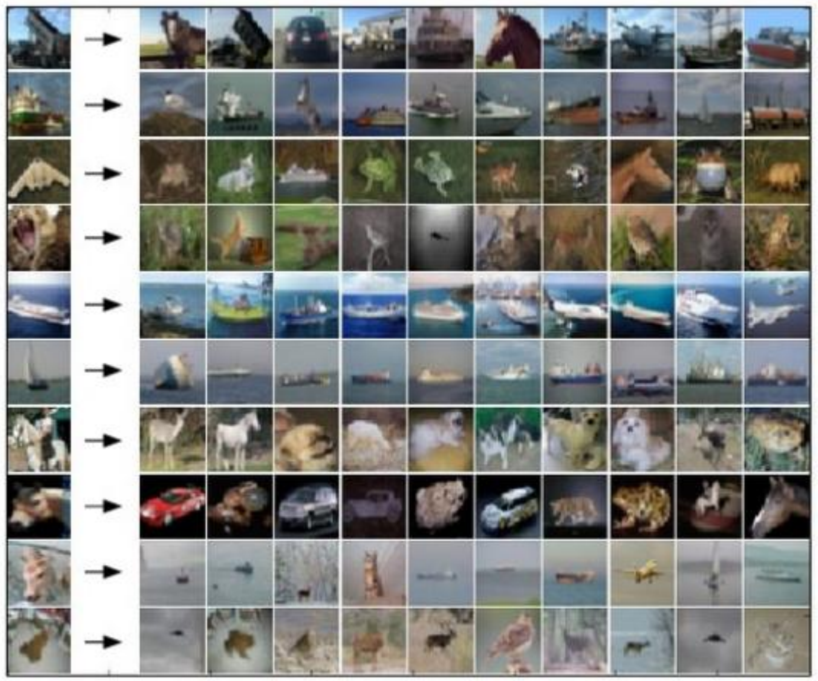
An example training set for four visual categories. In practice we may have thousands of categories and hundreds of thousands of images for each category

The image classification pipeline

- Input
  - Training set that consists of a set of *N* images, each labeled with one of *K* different classes
- Learning
  - What everyone of the classes look like. *training a classifier*, or *learning a model*
- Evaluation
  - we evaluate the quality of the classifier by asking it to predict labels for a new set of images that it has never seen before

Nearest Neighbor Classifier



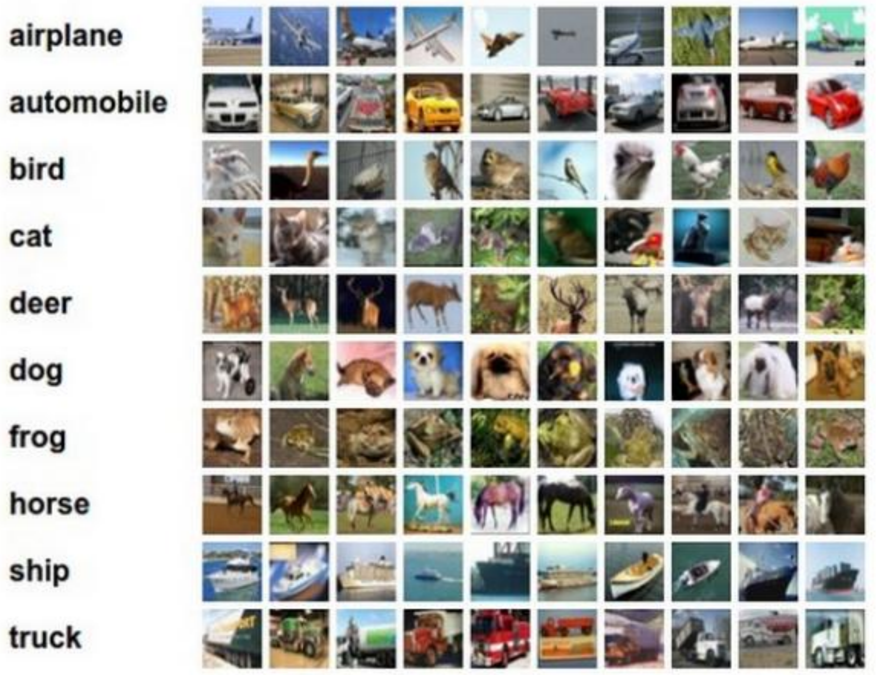


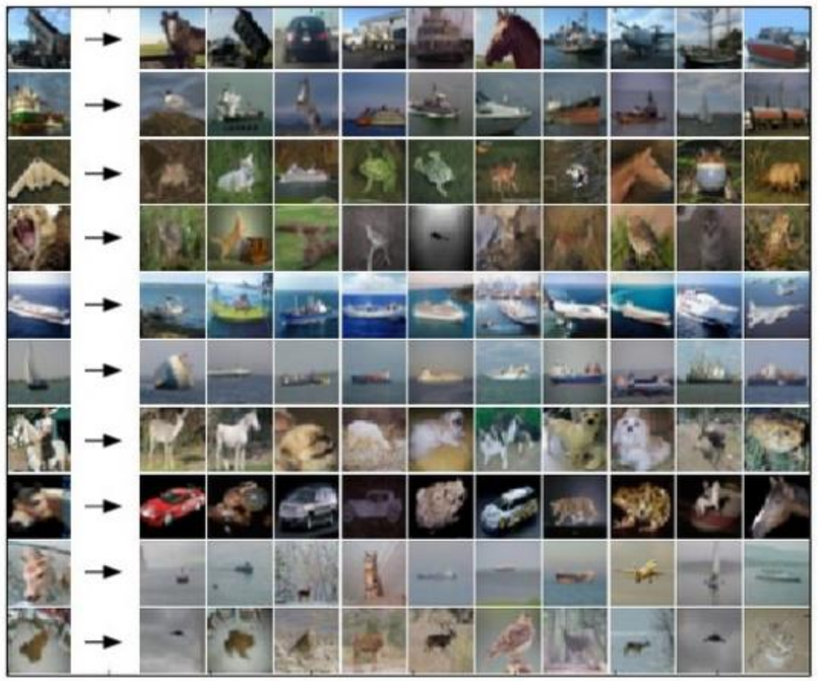
Left: Example images from the CIFAR-10 dataset. Right: first column shows a few test images and next to each we show the top 10 nearest neighbors in the training set according to pixel-wise difference.

L1 distance: $$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

Nearest neighbor accuracy: 38.6%

The choice of distance





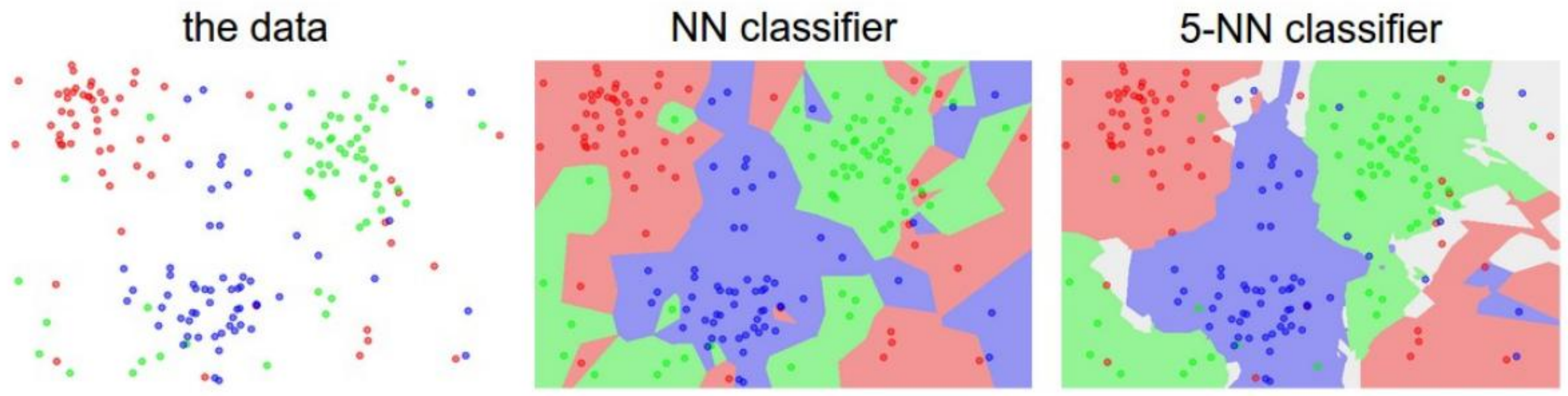
Left: Example images from the CIFAR-10 dataset. Right: first column shows a few test images and next to each we show the top 10 nearest neighbors in the training set according to pixel-wise difference.

L2 distance: $d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$

Nearest neighbor accuracy: 35.4%

k - Nearest Neighbor Classifier

- Instead of finding the single closest image in the training set, we will find the top k closest images, and have them vote on the label of the test image



L2 distance

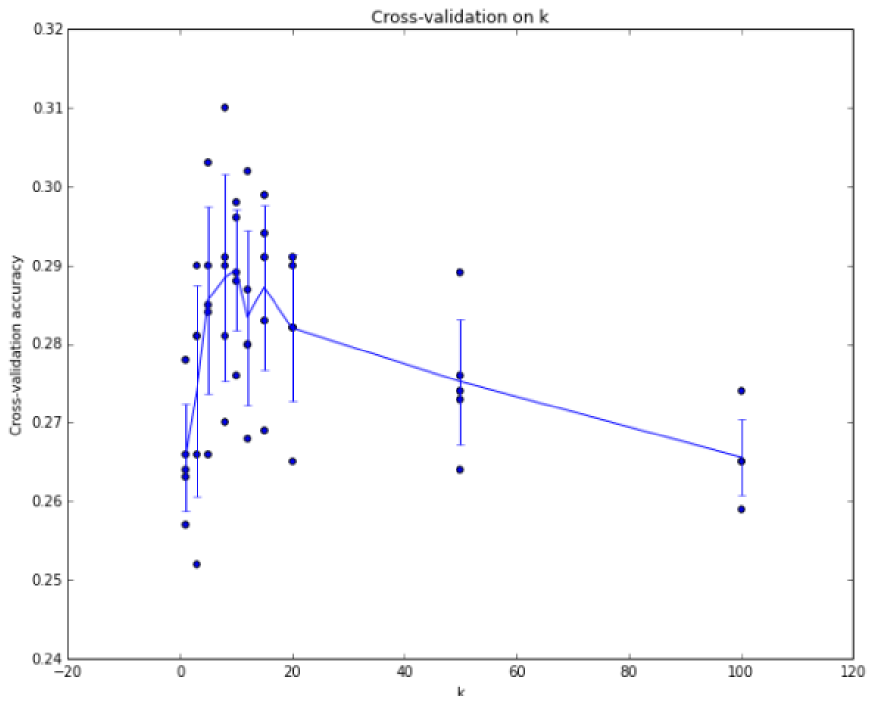
What are your observations?

Validation sets for Hyperparameter tuning

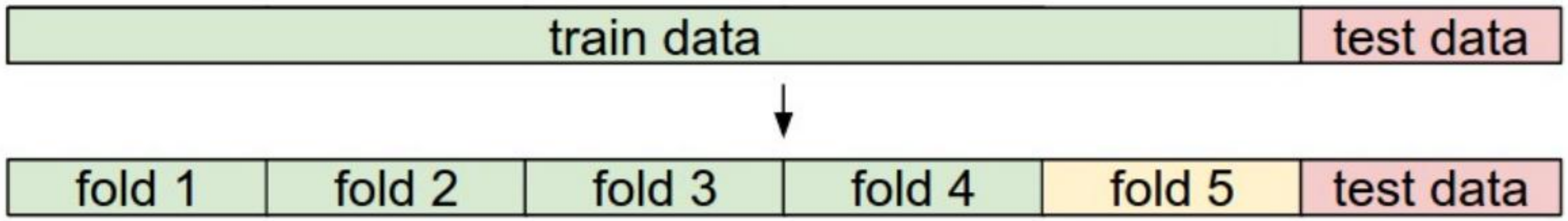
- **hyperparameters** (L1/L2, k) come up very often in the design of many Machine Learning algorithms that learn from data. not obvious what values/settings to choose.

- Cross-validation (Many choices): in 5-fold cross-validation, we would split the training data into 5 equal folds, use 4 of them for training, and 1 for validation.

5-fold cross-validation



Example of a 5-fold cross-validation run for the parameter **k**. For each value of **k** we train on 4 folds and evaluate on the 5th. Hence, for each **k** we receive 5 accuracies on the validation fold (accuracy is the y-axis, each result is a point). The trend line is drawn through the average of the results for each **k** and the error bars indicate the standard deviation. Note that in this particular case, the cross-validation suggests that a value of about **k** = 7 works best on this particular dataset (corresponding to the peak in the plot). If we used more than 5 folds, we might expect to see a smoother (i.e. less noisy) curve.



Pros and Cons of Nearest Neighbor classifier

- Very simple to implement and understand

- Computationally expensive
  - Approximate nearest neighbor using spatial data structures

- Requires a good distance metric (learned one)

Linear Classifier

- Stepping stone for neural networks
- Score function
  - that maps the raw data to class scores
- Loss function
  - that quantifies the agreement between the predicted scores and the ground truth labels

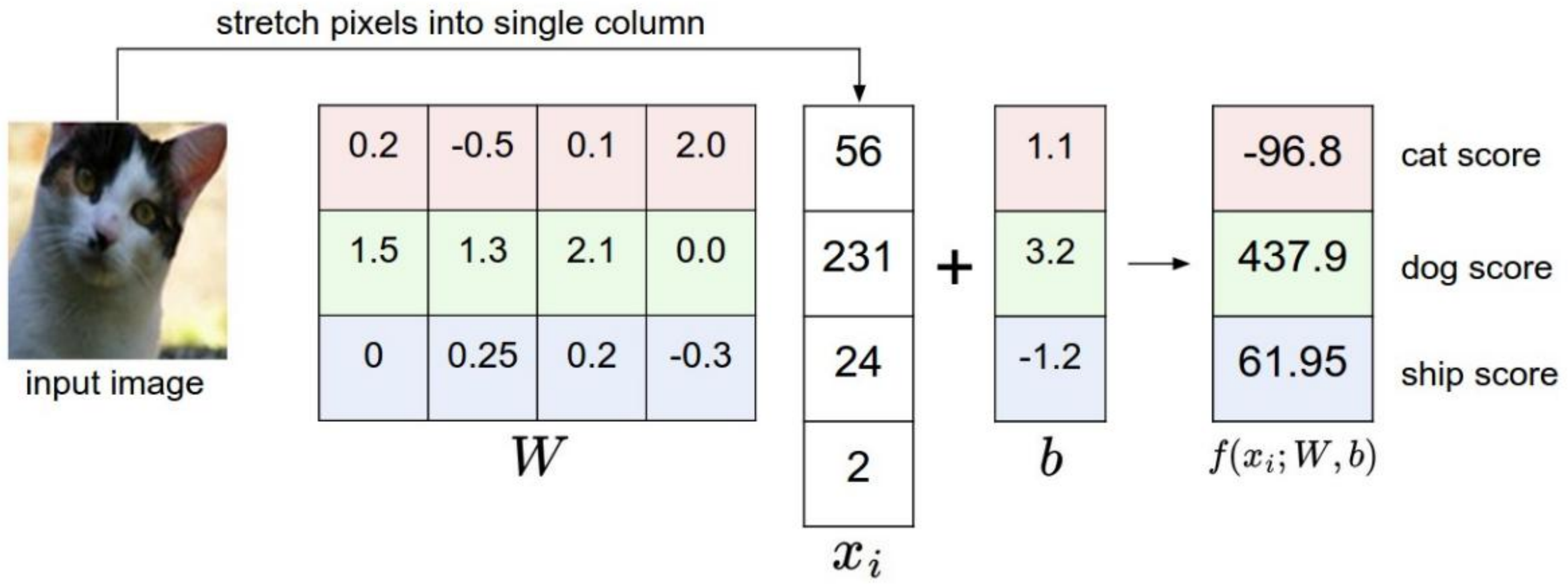
Parameterized mapping from images to label scores

- We will start out with arguably the simplest possible function, a linear mapping:

$$f(x_i, W, b) = \underbrace{W}_{\textbf{weights}} x_i + \underbrace{b}_{\textbf{bias vector}}$$

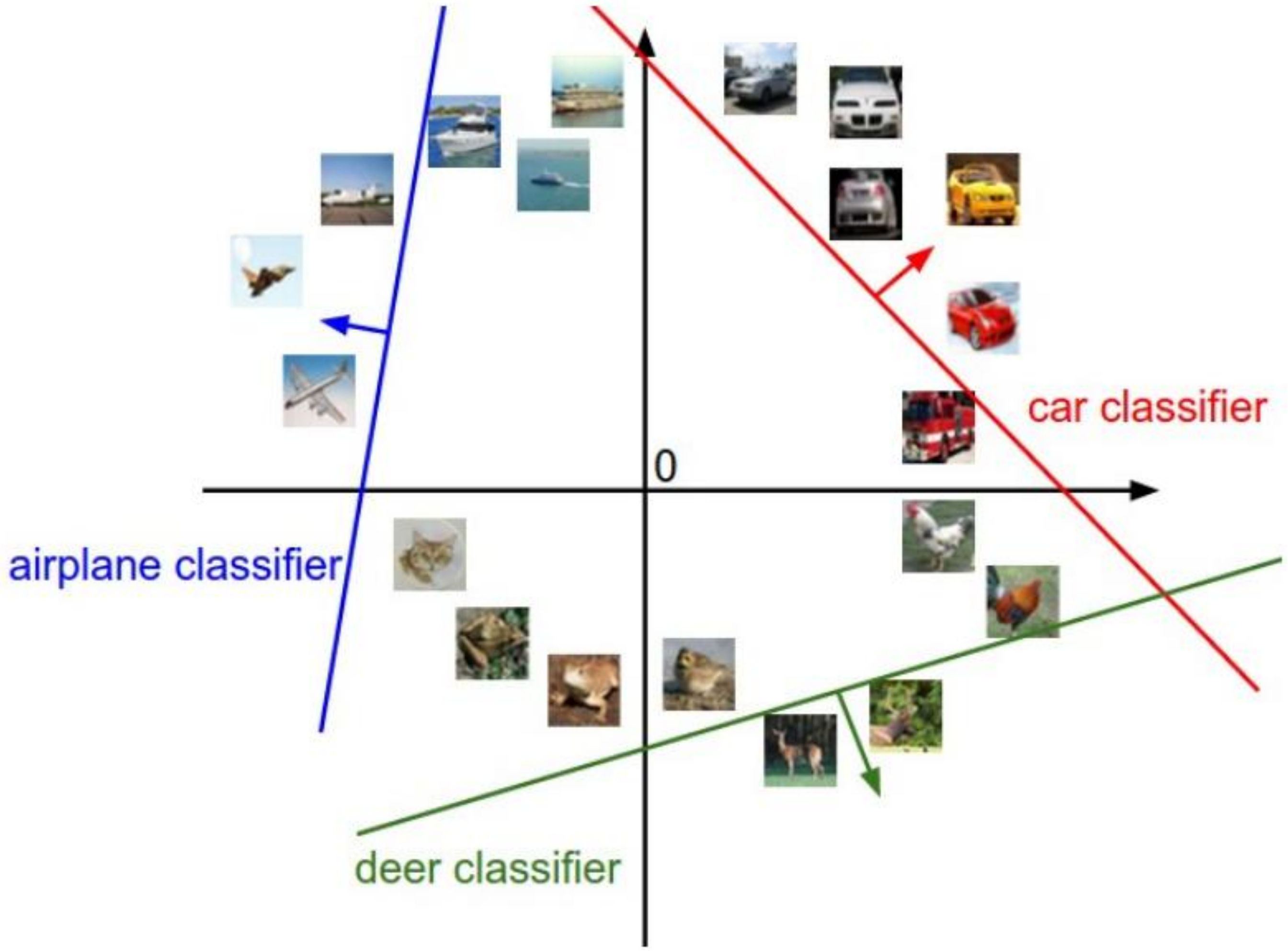
- the single matrix multiplication Wxi is effectively evaluating 10 separate classifiers in parallel (one for each class), where each classifier is a row of **W**
- once the learning is complete we can discard the entire training set and only keep the learned parameters
- classifying the test image involves a single matrix multiplication and addition, which is significantly faster than comparing a test image to all training images

Interpreting a linear classifier



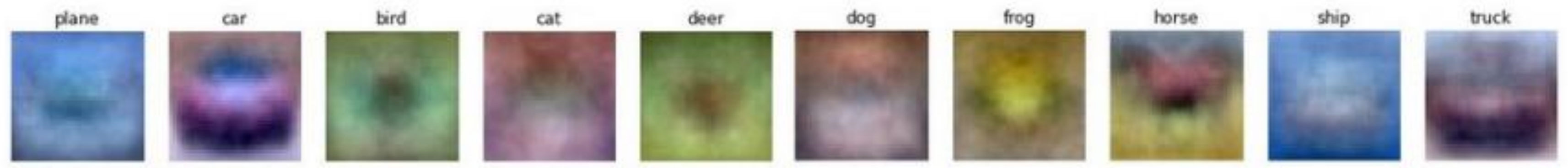
An example of mapping an image to class scores. For the sake of visualization, we assume the image only has 4 pixels (4 monochrome pixels, we are not considering color channels in this example for brevity), and that we have 3 classes (red (cat), green (dog), blue (ship) class). (Clarification: in particular, the colors here simply indicate 3 classes and are not related to the RGB channels.) We stretch the image pixels into a column and perform matrix multiplication to get the scores for each class. Note that this particular set of weights W is not good at all: the weights assign our cat image a very low cat score. In particular, this set of weights seems convinced that it's looking at a dog.

Analogy of images as high-dimensional points



Cartoon representation of the image space, where each image is a single point, and three classifiers are visualized. Using the example of the car classifier (in red), the red line shows all points in the space that get a score of zero for the car class. The red arrow shows the direction of increase, so all points to the right of the red line have positive (and linearly increasing) scores, and all points to the left have a negative (and linearly decreasing) scores.

Interpretation of linear classifiers as template matching



Skipping ahead a bit: Example learned weights at the end of learning for CIFAR-10. Note that, for example, the ship template contains a lot of blue pixels as expected. This template will therefore give a high score once it is matched against images of ships on the ocean with an inner product.

The score of each class for an image is then obtained by comparing each template with the image using an *inner product* (or *dot product*) one by one to find the one that "fits" best. With this terminology, the linear classifier is doing template matching, where the templates are learned.

Bias trick

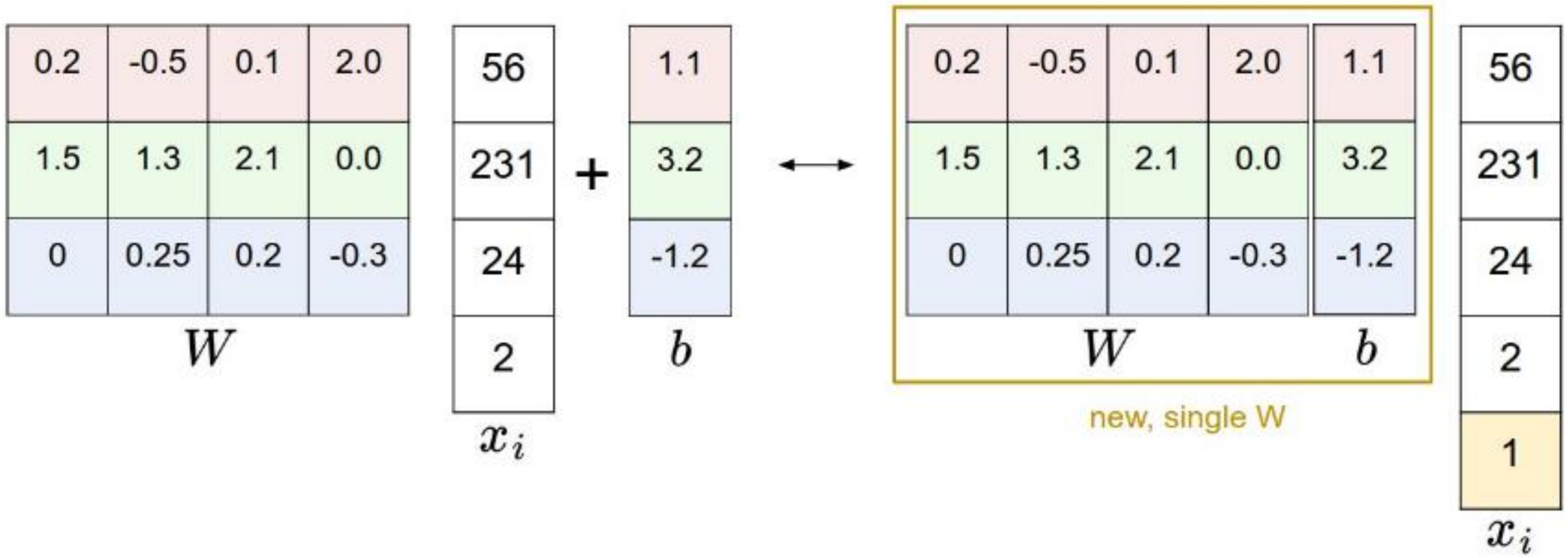


Illustration of the bias trick. Doing a matrix multiplication and then adding a bias vector (left) is equivalent to adding a bias dimension with a constant of 1 to all input vectors and extending the weight matrix by 1 column - a bias column (right). Thus, if we preprocess our data by appending ones to all vectors we only have to learn a single matrix of weights instead of two matrices that hold the weights and the biases.

Loss Term

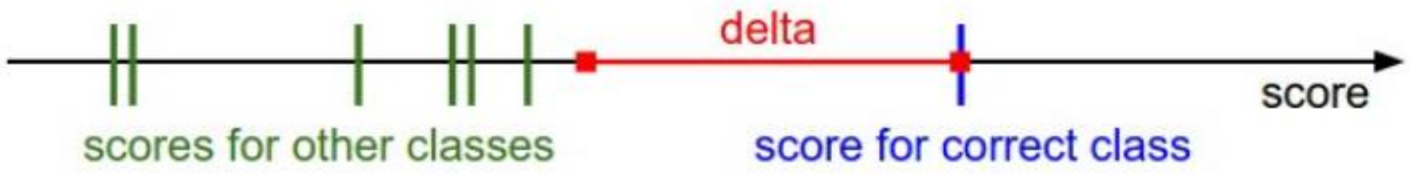
- Multiclass Support Vector Machine loss
  - The score for the j-th class is the j-th element: $s_j = f(x_i, W)_j$.

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)$$

  - For linear score functions

$$L_i = \sum_{j \neq y_i} \max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta)$$

Hinge loss: $max(0, -)$



Regularization term

- If W induces a zero loss, then λW where $\lambda > 1$ will also induce a zero loss
- Regularization penalty

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

- Full multiclass SVM loss becomes

$$L = \underbrace{\frac{1}{N} \sum_i L_i}_{\text{data loss}} + \underbrace{\lambda R(W)}_{\text{regularization loss}}$$

- Full form

$$L = \frac{1}{N} \sum_i \sum_{j \neq y_i} \left[\max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + \Delta)\right] + \lambda \sum_k \sum_l W_{k,l}^2$$

Softmax classifier

- It turns out that the SVM is one of two commonly seen classifiers. The other popular choice is the **Softmax classifier**, which has a different loss function
- **Cross-entropy loss** that has the form:

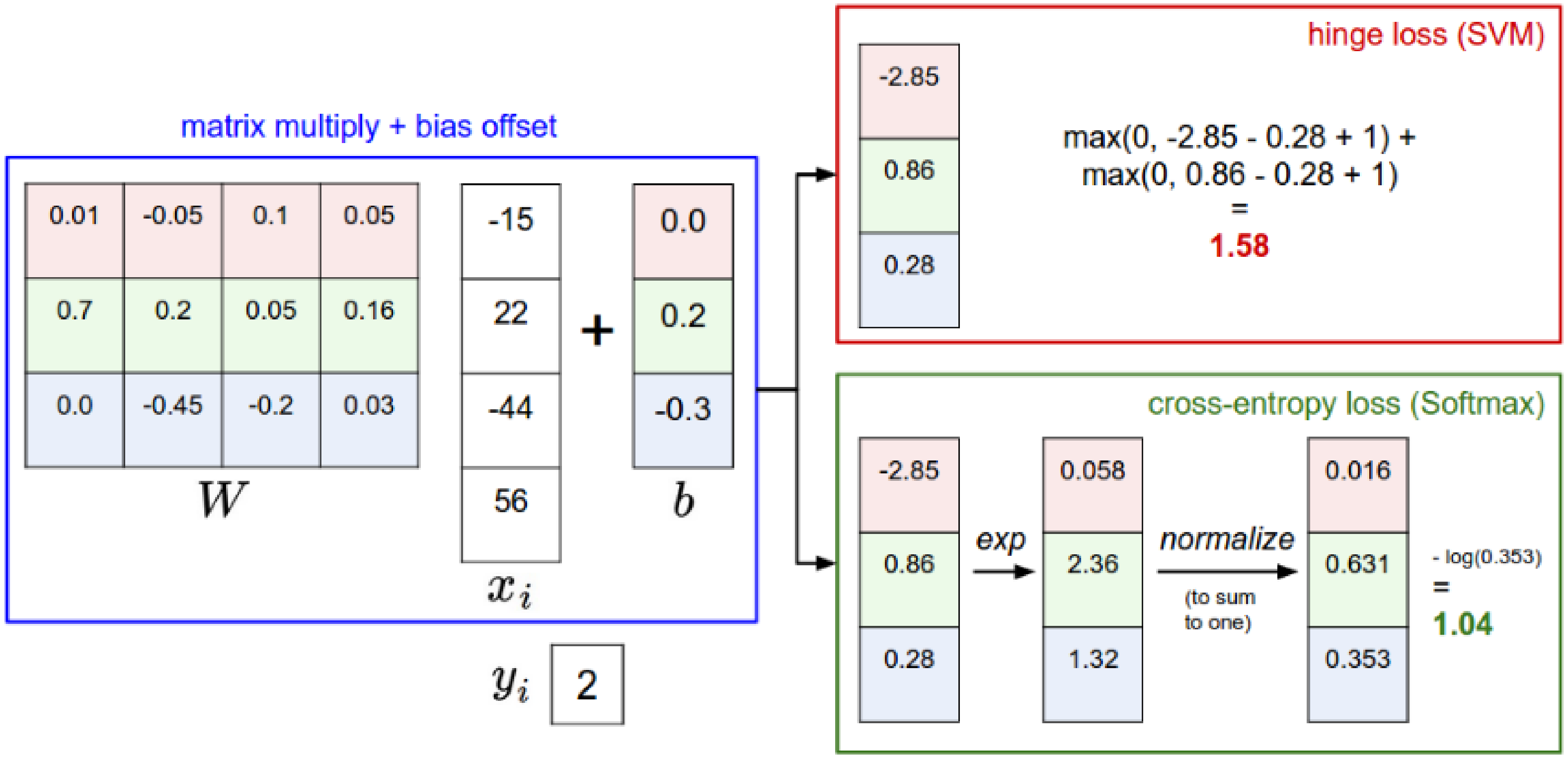
$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right) \qquad \text{or equivalently} \qquad L_i = -f_{y_i} + \log\sum_j e^{f_j}$$

- **Information-theory view**

$$H(p,q) = -\sum_x p(x)\log q(x)$$

The Softmax classifier is hence minimizing the cross-entropy between the estimated class probabilities ($q = e^{f_{y_i}} / \sum_j e^{f_j}$ as seen above) and the "true" distribution, which in this interpretation is the distribution where all probability mass is on the correct class (i.e. $p = [0, \ldots 1, \ldots, 0]$ contains a single 1 at the y_i -th position.).

SVM vs. Softmax



Softmax classifier provides “probabilities” for each class

In practice, SVM and Softmax are usually comparable

Summary

- K-nearest neighbor classifier – non parametric
- Support vector machine – parametric
- Softmax classifier - parametric